剛体円盤分子動力学シミュレーションにおける大規模計算と高速化の手法

AUTHOR(S):

礒部, 雅晴

CITATION:

礒部, 雅晴. 剛体円盤分子動力学シミュレーションにおける大規模計算と高速化の手法. 物性研究 1999, 72(1): 21-41

ISSUE DATE:

1999-04-20

URL:

http://hdl.handle.net/2433/96587

剛体円盤分子動力学シミュレーションにおける大規模計算と高速化の手法

九州大学理学研究科物理学専攻, 礒部雅晴[1]

(1998年11月30日受理)

Event-Driven 法による剛体円盤系の分子動力学シミュレーションの単純かつ汎用な高速アルゴリズムを開発した. このアルゴリズムによる1イベント当りの計算量は粒子数 N に対して $\log N$ 程度であり, その係数も旧来の最も速いといわれているアルゴリズムより小さい. 実際に, 2500粒子系のシミュレーションを行い, Alpha500 互換機で約 50 Million Collision Events/CPU Hour の計算速度を実現した. また, 今回開発したアルゴリズムの無限空間系への拡張も提案する.

1　はじめに

コンピューターの演算能力の向上により, 90年代に入ると「グランドチャレンジング問題」と呼ばれる科学技術上の様々な大規模問題を現実的に解くという段階に入ってきた.

ところが大規模問題の場合, いくらコンピューターというハードウェアの進歩があっても, その進歩には限界があり, 実際にシミュレーションをするにはその問題に適したアルゴリズムすなわちソフトウェアの開発が不可欠となる. たとえば, 系を構成する要素 N に対して, 演算量が N^2 で増えていくようなアルゴリズムは大規模計算に適さない. よって大規模シミュレーションを行う際, 最も注意を払う点はアルゴリズムの演算量をいかにして減らすかということであり, それが重要なポイントとなる.

分子動力学法 (Molecular Dynamics, 以下 MD と略す) は, Alder と Wainwright により, 1957年に初めて論文 [1, 2] で発表され, それ以降の一連の仕事がある [3, 4, 5, 6, 7]. Alder と Wainwright は, 粒子を剛体円盤 (3次元では剛体球) としてシミュレーションを行い液体-固体の相転移点近傍では, このような反発力しかない粒子系でも結晶化し固液相転移を起こすという, 当時の常識を打ち破る事実を発見した. これは後のコンピューターシミュレーションの発展に大きな影響力を与えることとなった.

剛体円盤系では, 各粒子間の衝突する時間を除けば各粒子は直線運動をし, よってダイナミクスは衝突と直線運動の2つだけである. これは, 時間的に離散的なイベントが次々と起こっていると見なすことができるので, 時間刻みを設定して微分方程式 (Newton の運動方程式) を解く必要がない. 時間刻みを設定して逐次差分方程式を解いていく一般によく知られてた MD を Time-Step-Driven 型 MD (TSDMD) と呼ぶことがある. これに対し, 剛体円盤系のようにイベントを単位にしてシミュレーションを進めていくタイプの MD シミュレーションを Event-Driven 型 MD (EDMD) と呼ぶ.

剛体円盤系は, 同様に反発力のみを持つ短距離相互作用するソフトコア系の MD シミュレーションと比べるとアルゴリズムが全く異質であり, 高速化には複雑で高度なアルゴリズムとデータ構造の知識が必要である.

剛体円盤系の高速化のポイントは未来に起こるイベントのキューを考慮して, そのデータ構造をいかにうまく取り扱うかである.

[1] E-mail: isobe@stat.phys.kyushu-u.ac.jp

剛体円盤系の大規模計算を高速化するアルゴリズムの革新的なブレークスルーは，Rapaport (1980) [15] によってなされた．詳細は 3 節に譲るが，Rapaport の戦略は空間を小さな長方形のセルに分割する Subcell Method を用い，イベントを粒子間の衝突による Collision Event と粒子がサブセルを横切るという Cell-Crossing Event の 2 つとするものである．またそれぞれのイベントの起こる時間を節 (node) として，2 分探索木 (binary search tree) を作り，最も短い時間で起こるイベントを探索する計算量を $\mathcal{O}(\log N)$ にした．しかし，Rapaport のアルゴリズムは非常に複雑でわかりずらく，コーディングの際に大きな困難を伴うのが欠点であった．そこで 90 年代に入ると，大規模計算を行うため，Rapaport の戦略を発展させデータ構造をより単純にし効率性を上げるといういくつかのアルゴリズムの提案があった [16, 17, 19, 20, 21]．

上にあげた Rapaport 系列のアルゴリズムの最大の特徴は，Cell-Crossing Event を考慮することである．しかし，Cell-Crossing Event 自体は単に仮想的に作ったサブセルを横切るというものであるので，物理的な意味は全くない．よって，ある一定衝突回数のシミュレーションをする際は，Cell-Crossing Event は単なるオーバーコストであり，無駄な計算である．また，Cell-Crossing Event は Collision Event に比例して増大するという性質を持つ．

そこで筆者は，Rapaport の戦略（すなわち Cell-Crossing Event を考える）とは異なる戦略を基にしたアルゴリズムを提案した．それは，Form et al.[13] によって，TSDMD の短距離相互作用する系において開発された Exclusive Particle Grid Method に着目したもので，筆者はこのアイデアを剛体円盤系の EDMD に適用し，その拡張を試みた．そしてその計算量のオーダーを，Rapaport 系列のものと理論的に比較し考察した．その結果，Cell-Crossing Event に対応する部分の計算量オーダーがわずかではあるが，Rapaport 系列のそれよりも優れている，すなわち高速であることを示すことができた．また実際に筆者の提案した戦略を基にしてに 2 次元剛体円盤 MD のプログラムをコーディングし，シミュレーションを行ったところ，それが過去の文献で公表されている効率性よりもパフォーマンスのよいことがわかった．

Cell-Crossing Event を考える Rapaport 系列のアルゴリズムに比べると筆者のアルゴリズムは，通常の TSDMD でよく使われている高速化の手法，Linked-Cell Method と Neighbour List の概念を拡張したものである．これは適度に単純で TSDMD をしている方にも受け入れやすく，剛体円盤系でのアルゴリズムを高速化する際のコーディングの困難を緩和するものと思われる．

ソフトコア系，剛体円盤系を問わず，これまで無限に広がった系での MD シミュレーションは，Conventional Subcell Method に基づいた高速化は不可能と思われてきた [17]．その最大の原因は，系が空間が無限に広がっているがために，空間を小さなセル分割する際，無限のサブセルを用意しなくてはならず，これはすなわち無限のサブセルのための配列を確保しなければならないことを意味し，一方でコンピューターのメモリ資源は有限であるため，シミュレーションは原理的に不可能となるからである．

しかしサブセルの多くは粒子を 1 つも含んでいないことに注目すると，実際に粒子の入っているセルのみメモリ上に配列を確保してもセルの位置と粒子番号の情報が得られるので，これはなんらかのアルゴリズム的な改良によりその困難を克服できる可能性が残されていることをほのめかしている．

筆者はこの問題に対し，無限のサブセルに必要な情報を有限の情報に圧縮するやり方を提案し，さらに，高速に隣接セルの情報を引き出すがために，データ探索で一般的に使われているハッシュ法を応用して改良を試みた結果，それが可能であることを示した．特に筆者が今回開発した Linked Cell Method の延長線上にある Extended Exclusive Particle Grid Method にはこの境界のない場合は比較的容易に適用可能であることがわかった．この方法の詳細と具体的な適用例は，8.2 節で示すことにする．

大規模EDMDが可能となれば, 外場の導入や衝突則を変えることで極めて広範囲な分野での応用が可能と思われる. これまでの代表的な研究の一例を上げると, 平衡系相転移（固体-液体転移, 2次元融解）[1, 2, 3, 4, 5, 6, 7, 47], 非平衡流体系 (Rayleigh-Bénard 対流系など）[24, 25, 26, 27, 28, 29, 30, 31, 48], 非平衡化学反応系 [50], 非平衡散逸系（粉体系）[32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42], ランダムディスクパッキングの問題 [44, 45], ガラス系 [43], ポリマー系 [46, 22], 土星の輪の形成過程 [17] などがEDMDで行われてきたが, 高速なアルゴリズムと計算機の進歩により, これからさらに大規模な計算が可能になると思われ, 剛体円盤系のシミュレーションの利用価値はこれからもますます増していくと期待される.

本論文の構成は, 以下の通りである. まず, 2節で剛体円盤分子動力学シミュレーションに必要な基本的な式を紹介する. 3節では, 過去の文献で開発された高速化の手法を系統的に紹介する. 4節, 5節では, 今回開発した手法に関して紹介し, 6節では, この開発した手法とRapaport系列の戦略の計算量を理論的に解析する. 7節では, 実際にシミュレーションした時の場合の計算時間を過去の文献と比較し, 8節では, 新たに開発したアルゴリズムの適用範囲を調べるため, 多分散系と無限に広がった系への適用について説明する. 最後に, 9節でまとめる.

なお, 以下の節では, 簡略化のため2次元系すなわち, 剛体円盤系に絞って話を進めるが, 3次元系（剛体球系）への拡張は容易に行うことが出来ることを付け加えておく.

2 剛体円盤分子動力学シミュレーション

剛体円盤系シミュレーションの特徴は, 粒子速度の変化は衝突の瞬間だけ起こり, その瞬間を除いては速度は変化しないということである. 衝突前後の速度は, 衝突前の2個の剛体円盤の速度と, その衝突時の相対的位置だけで決まる.

2.1 剛体円盤のポテンシャル関数

以下に剛体円盤のポテンシャルを示す.

$$\phi(r) = \begin{cases} \infty & r \leq \sigma_{ij} \\ 0 & r > \sigma_{ij} \end{cases} \tag{1}$$

$$(\sigma_{ij} = \sigma_i + \sigma_j)$$

σ_i は, 粒子 i の半径を表す. 剛体円盤 i, j は, 粒子間距離 r_{ij} が σ_{ij} になったとき衝突し, それ以上の距離では等速直線運動をする.

これは, 非現実的なポテンシャルかもしれないが, その簡単さゆえによく用いられる. また, 解析的な理論から得られる解の吟味に用いたり, 摂動理論の出発点として使われる.

2.2 衝突に関する基本的な方程式 (Collision Event の計算)

剛体円盤の運動は, 常微分方程式を数値的に解く必要はなく, 位置と速度の時間発展は代数方程式を解くことによりもとまる. ここでは, 衝突前の粒子 i, j の速度を v_i, v_j とし, 時刻 t_c で衝突が起こり, 衝突後の速度を v_i', v_j' とする.

まず, 位置座標の時間発展式は衝突が起こらない限り粒子は等速直線運動をするので, 位置座標 $\mathbf{r}_i(t)$ は, 次の式で書き表すことができる.

$$\mathbf{r}_i(t) = \mathbf{r}_i(0) + \mathbf{v}_i t \tag{2}$$

$\mathbf{r}_i(0)$ は, 前回衝突したときの座標で, t は, 前回衝突したときから現在までの時間である. このとき, $\mathbf{v}_i$ は一定の値をとる. よって, 位置座標は, 次に衝突する時刻と衝突後の速度がわかれば決定できる.

次に, 衝突する時間の求め方であるが, 相対位置, 相対速度をそれぞれ, $\mathbf{r}_{ij} = \mathbf{r}_i(0) - \mathbf{r}_j(0)$, $\mathbf{v}_{ij} = \mathbf{v}_i - \mathbf{v}_j$ とし, $b_{ij} = \mathbf{r}_{ij} \cdot \mathbf{v}_{ij}$ を定義する. 衝突の瞬間は次の式が成り立っているので,

$$|\mathbf{r}_i(t_c) - \mathbf{r}_j(t_c)| = \sigma_i + \sigma_j \tag{3}$$

(2) 式と (3) 式から,

$$t_c = \frac{-b_{ij} - \sqrt{b_{ij}^2 - \mathbf{v}_{ij}^2(\mathbf{r}_{ij}^2 - \sigma_{ij}^2)}}{\mathbf{v}_{ij}^2} \tag{4}$$

が導き出せる.

衝突後の速度は, 運動量, エネルギー保存の法則を使って次の式が導き出せる. これを衝突則 (Collision Rule) という.

$$\mathbf{v}_i' = \mathbf{v}_i - \frac{2b_{ij}}{\sigma_{ij}^2}\left(\frac{m_j}{m_i + m_j}\right)\mathbf{r}_{ij} \tag{5}$$

$$\mathbf{v}_j' = \mathbf{v}_j - \frac{2b_{ij}}{\sigma_{ij}^2}\left(\frac{m_i}{m_i + m_j}\right)\mathbf{r}_{ij} \tag{6}$$

粉体系のような散逸のある系の場合には, この衝突則を反発係数 r を使って書きなおせばよいだけである.

このように剛体円盤系のダイナミクスは極めて単純明快なものである. シミュレーションの際は, 系のすべて粒子ペアの衝突時間を計算し, 最も短いものを探して系全体をその時間だけ進める. これを繰り返すことにより, 原理的にはシミュレーションは可能である. しかしこのやり方は, 粒子数の増大に伴い計算量が $\mathcal{O}(N^2)$ で増大していき, 大規模計算するのは適さないことがすぐにわかる.

そこで, いかにしてこのプロセスを演算量を少なくしていくかが高速化のポイントとなるのである.

3　高速化の 4 つのカテゴリー

この節では, 過去に提案された様々な高速化のためのアルゴリズムの手法を系統的に紹介する. EDMD シミュレーションの高速化のテクニックは, 次の 4 つのカテゴリーに分類される.

3.1　Subcell Method, Linked Cell Method, Neighbour List

まずは, いかにして考慮する粒子ペアの数を少なくするかである. 粒子ペアはお互いの距離が短い程, 衝突時間が短くなるわけで, この性質に着目して考慮する粒子ペアはごく近傍の粒子のみを考えるのが妥当であることがわかる. 粒子ペアの数が削減できればその結果, 計算すべき粒子間の

衝突時間の量が削減でき, 効率性が増す. これは, 1 粒子当たりに計算すべきイベントを出来るだけ少なくすることに対応する.

Erpenbeck et al.[8] は, EDMD シミュレーションを行う際に, 系を小さなセルに分割するやり方を提案した (Subcell Method). この時, 小さなセルの一辺の長さは, 粒子の直径よりも大きくとる. 系をサブセルに分割した際, 粒子ペアは分割したセルに所属している粒子と隣接するセル（2 次元では 8 つ）に所属する粒子のみを考慮するとする (Linked Cell Method (LCM)[10, 11]). このセルに分割するやり方は, 計算量が $\mathcal{O}(N^2)$ から $\mathcal{O}(N)$ となるため, 分子動力学法を使って大規模計算をする際, 効率性を上げるための常套手段としてよく使われる手法である.

Erpenbeck et al.[8], Rapaport[15] は, EDMD シミュレーションをする際, この Subcell Method に基づいてアルゴリズムを発展させた. それは, 系をセルに分割して LCM により Link List を製作して, それに登録されている粒子ペアを考慮して衝突する時間を計算するだけでなく, 今いるサブセルから隣のサブセルへ移るまでの時間も計算しておき, それをイベントとして見なそうというアイデアである. よってイベントは, Collision Event と Cell-Crossing Event の 2 つとなる. Cell-Crossing Event が起こった場合は, イベントが起こる前に所属していたサブセルとの境界を除き, イベントが起こった後に所属することになるサブセルに隣接するサブセル (3 つ) との境界を横切るまでの時間を計算し, それをイベントとする. EDMD の高速化は上記のやり方が標準的であり, このアイデアを基にして最近のアルゴリズム開発も行われている [16, 17, 19, 20, 21].

一方で, Verlet(1967)[9] により提案された Neighbour List (NL) を EDMD に応用したやり方も考えられる [50].

Smith et al.(1997)[22] は, TSDMD の高速化の手法で一般に用いられている LCM + NL を用いてポリマー系の計算をしておりこれもまた高い効率性を得ている.

筆者はこの LCM + NL の戦略を基に, それぞれ拡張を試みた (4 節, 5 節).

3.2 Event List

この節では, 説明をわかりやすくするため, 前節で述べた Subcell Method を使わない場合を考える. 2 節の終わりでも述べたように, 各イベント毎にすべての粒子ペアの衝突時間を計算していては, $N(N-1)/2$ 回の衝突時間の計算を行わなくてはならない. それでは計算量が $\mathcal{O}(N^2)$ となり非効率的で大規模計算に適さないことがすぐにわかる. この困難を克服するために, 粒子ペアの未来に起こる衝突時間についてあらかじめリストに保存しておくことにする. そうすると, 次のイベントを計算する際には衝突に関与した粒子のみ衝突時間を計算しなおし, その他はリストに保存したものを使えば再計算の手間が省けるので計算量が $\mathcal{O}(N)$ となり, 効率的な計算が実行可能となる. このような演算効率を上げるためにリストを作る方法は, 歴史的に見ても早くから用いられていた方法である [3].

リストに衝突時間を保存する際には, すべてのイベントを保存しておきリストを作成するやり方が考えられる. すなわち粒子 i に対し, 衝突する可能性のある粒子 j が N_j 個あるという場合, $N \times N_j$ の大きさの配列にイベントが保存される. このリストを Multiple-Event Time List と呼ぶことにする. Multiple-Event Time List を使うと, N_j があらかじめわからないため、Event List のサイズの評価をしなくてはいけない.

Multiple-Event Time List は, イベントを進めるたびに更新されるものである. この更新はなるべく最小にしたい. Event List をうまく削減することは, 大幅な計算量の削減されることになる.

Event List の大きさを削減する方法としては, 志田ら [17] は $2N$ の大きさのリストに削減するやり方を提案した. また, Lubachevsky[16] は, 各粒子 i に関与する最も小さな衝突時間のみを Event

List に保存しておくというやり方を提案し，リストの大きさ N に削減した．Allen et al.[14] も同様のやり方をその著書のプログラムで表現している．このリストのことを Single-Event Time List と呼ぶことにする．Single-Event Time List は Multiple-Event Time List に比べて，大幅なメモリの節約となる．また，このようにデータ構造を単純なものにすると，3.3 節で述べる Event List Scheduling を容易にし，効率性が増すというメリットがある．

さて，各粒子に関連した最小の衝突時間だけを Event List に残すという Single-Event Time List のやり方は実は情報を削減しすぎており，Event List の更新の際，衝突に関連した粒子だけではなく，さらに別の粒子に対してもイベント時間の再計算が必要であるという欠点がある．

Marín et al.[19] は，Event List に対して Local Minima Algorithm(LMA) という手法を提案した．これは，Multiple-Event Time List を設けておき，実際に Event List Scheduling する際には，各粒子 i に関する最小のイベントのみをキューに入れるというやり方である．彼らの論文では，上の 2 つのやり方と彼らが提案した LMA が比較され，LMA が演算効率の点で優れていることが示されている．

なお，前節で述べた Subcell Method を使って，Event List を製作する際は，Cell-Crossing Event も Collision Event と対等に扱いリストに保存する．

3.3 Event List Scheduling

さて，Event List が作成できたら次は，いかにしてこのリストから最も短い時間のイベントを効率よく探索するかである．最も単純なやり方は，リスト内の時間を端から 1 つずつ比較していくやり方である．これは線形探索法と呼ばれ，計算量は $\mathcal{O}(N)$ となる．

1980 年，Rapaport[15] は，最小時間の探索方法として効率のよい方法として知られていた 2 分探索木 (Binary Search Tree: BST)[23] を用いるというブレークスルーにより，これまでのシミュレーションよりも大幅な効率性を持つアルゴリズムの開発に成功した．Rapaport は Subcell Method を使い，Collision Event と Cell-Crossing Event を考慮して Multiple-Event Time List を作成し，それぞれのリストの要素を節 (node) として，BST を作成した．イベントが起こったときは，それぞれ Collision Event や Cell-Crossing Event に関連した粒子ペアもしくは粒子とセルについての 2 つの Circular List をたよりにイベントに関連した節の削除を行い，イベントに関連した粒子に対しては再計算を行って新たな節を作り，それらを BST に挿入することによって BST を作り直し，最小時間の探索を行う (BST の最も左にある節が最小時間のイベントである)．このやり方により，1 イベントにかかる計算量は平均で $\mathcal{O}(\log N)$ となり[2]，大幅な計算量の削減を実現した．

しかし，Rapaport のアルゴリズムは非常に複雑であり，1 つの節には，注目した粒子の番号と衝突する粒子の番号もしくは Cell-Crossing するサブセルの記号を保存する整数配列，BST の節をつなぐ 3 つのポインタ整数配列と，不要になったイベントを削除するために必要な Circular List の 4 つのポインタ整数配列，さらにはイベントの起こる時間を保存する実数配列を必要とする．

イベントの後は，BST の削除に伴う節の枝のつなぎ変えなどが煩雑なプロセスを伴い，アルゴリズムの理解ならびにコード作成には大変な困難を要し，EDMD シミュレーションを使った研究の遂行に大きな障壁となったことが想像に難くない．

Rapaport のアルゴリズムの提案から 10 年以上後に，Lubachevsky[16] は，BST の代わりに Implicit Heap を使う方法を提案した．

最近 (1995 年) になって Marín and Corderc[20] は，$\mathcal{O}(\log N)$ で，高速に探索を行う様々なアルゴリズム (Complete Binary Tree, Implicit Heap, Binary Search Tree, Pairing Heap, Skew Heap,

[2] 計算量に関する議論では，log の底はつねに 2 である．

Splay Tree, Binary Priority Queue, Leftist Tree, Binomial Queue, Priority Tree) に対して, 網羅的に実際に EDMD シミュレーションを行い, どれが最も剛体円盤系に適したアルゴリズムであるかを調べた. その結果, 完全 2 分探索木 (Complete Binary Tree: CBT) がすべての密度領域で最も効率がよく, また粒子数増大に伴いほかの探索アルゴリズムより効率性がますます増すことを明快に示した.

CBT は, Event List を横に一列に並べスポーツ大会でお馴染みのトーナメントを行うやり方である. ここでは, 簡単のため Single-Event Time List を例にとり説明する. Event List が作成された時点で, すぐ隣のリストとイベント時間を比較して小さいほうが勝者となりさらに上の階層 (level) に勝ちあがることが出来る. これを繰り返すことにより, $N-1$ 回の対戦ののち最もイベント時間の小さいイベントが根 (root) に残り勝者が判明することになる (CBT 作成のための計算量は, $\mathcal{O}(N)$).

さてイベントが起こった後は, イベントに関連した粒子に対応する Event Time List に関してのみ変更されているため, 先に作った CBT が使える. よって, イベント時間の再計算が行われそれをリストに挿入しなおし, 再び最小時間イベントを探索するわけだが, 先に作った CBT を使ってリストの更新のあった部分のみ, 以前の勝者と対決しなおし, 新しい Event List が勝利する限りは根に向かって登りつめればよいわけである. よって, 計算量はいつも $\mathcal{O}(\log N)$ 以下ですんでしまう. 一方, Rapaport の BST は節のイベント時間が規則性を持っている場合など, 最悪 $\mathcal{O}(N)$ になる可能性もあり, こういった点が CBT に比べ効率性が落ちる理由と考えられる.

CBT には, もう一つ決定的な利点がある. それはリストから節を作る際に, 節の番号を保存する配列を新しく作る必要がないからである. あらかじめ, $2N-1$ の整数配列を用意しておく. 配列 N から $2N-1$ に粒子番号を並べておきそこに各粒子の最小時間を対応させるとする. 節 n は, $2n$ と $2n+1$ の子を持っており, すなわち n と $n+1$ の勝者が入る上の階層の配列番号は $[\frac{n}{2}]$ で表現できる. よって, 探索木をつなぐための無駄なポインタ配列を一切必要としないので, 大変なメモリの節約になる. Single-Event Time List を使う場合には, 節の挿入, 削除に伴う枝の付け替えも行う必要がなくアルゴリズムは大変単純化され, コーディングの困難も大幅に解消される.

Rapaport の BST と Marín and Cordero の CBT を比べると確かに理論的な計算量は $\mathcal{O}(\log N)$ で同じだが, 実際に EDMD シミュレーションする上では定数係数が違い, 単純さ, 効率性, メモリ容量すべての点で改善が施され, 大幅なアドバンテージがあるといえる.

3.4 Coordinate Updating

Event List の中で最小のイベント時間 t_{min} を探索できれば, 後は系のダイナミクスを t_{min} 進めるだけである. イベントに関連した粒子は, そのイベントに即した処理を施し, イベントに関連しなかった粒子は t_{min} だけ, 等速直線運動させればよい. しかしイベントに関連しなかった粒子は N が大きいと膨大になり, この部分の計算量は $\mathcal{O}(N)$ となってしまう. 局所的なイベントのため粒子全体の配置を変更するというやり方は非効率的であり, ここをいかにしてうまく処理するかが効率性を上げる際のポイントとなりうる.

Erpenbeck et al.(1977)[8] は, イベントに関与した粒子のみを処理し, 他の粒子はそのままにしておけば, 効率性を上げることができるというアイデアを基にアルゴリズムを提案した. Smith et al.(1997)[22] の文献では, False Positioning というアルゴリズムとして紹介されている. これは衝突の後に求めた速度を使って, 進めた配置から粒子のダイナミクスを逆転させて, 粒子配置を保存しておき粒子の時刻を常にある時刻 (例えば, $t=0$) に戻しておくというやり方である.

これと別のやり方として, Rapaport(1980)[15] は, 分割したサブセルに局所時間を割り当てる方

法を, また, Marín et al.(1993)[19] は, Delayed State Algorithm (DSA) という名前で, それぞれの粒子に固有の時間を設けて衝突に関連した粒子のみ時間を進めるというやり方を提案している. しかし, 局所時間を使うことは次の衝突時間を計算する時に注意を要するので気をつけなくてはならない. すなわち, 系の今の時間まで粒子を進めたときの配置から次の衝突時間を計算しなくてはいけないということである. 計算終了時には, 粒子毎の局所時間と系の今の時間の差の時間だけ各々粒子を進めて初めて正しい粒子配置を得ることが出来る.

False Positioningは効率性の点で余分な計算の分, Delayed State Algorithmにやや劣ると思われるが実際には全体の計算量にはほとんど影響せず, 効率は同じと見なせる. またFalse Positioningは, 各粒子毎に局所時間の配列を新たに用意する必要がなくなるためメモリ容量を節約できるという利点がある.

これらのやり方を使うと1イベント当たりのCoordinate Updatingにかかる計算量が, $\mathcal{O}(N)$ から $\mathcal{O}(1)$ になり大幅な効率化が実現できる.

4 Extended Exclusive Particle Grid Method

Event Listを製作並びに更新する際, Subcell Methodを使わない場合は, Event Listの製作には $N(N-1)/2$, 更新には $N-1$ の粒子ペアの衝突時間の計算が必要であることは既に述べた. しかし, 実際に衝突を起こす粒子ペアは配置が極めて近傍のものに限られており, 系全体の粒子ペアを考慮にいれるのは全くの無駄であり, それをなくすためSubcell Methodが使われる. Subcell Methodを用いるとEvent Listの製作, 更新にかかる計算量はそれぞれ, $\mathcal{O}(N)$, $\mathcal{O}(1)$ となる.

LCMは, 粒子直径よりも大きな一辺で系を分割するやり方である. しかし, サブセル内に何個粒子が入るかわからず, サブセル内粒子にはLink Listを作らなくてはならず, 実際にプログラムを作成してみるとわかるが多少複雑になる.

筆者は, 隣接粒子を最も効率よく探す方法として, ソフトコア(粉体)系MDで提案されたExclusive Particle Grid Methodに注目して, そのアイデアを拡張することにより高速化を試みた. この方法のアイデアの源泉は, Buchholtz and Pöschel (1993)[12], Form, Ito, and Kohring (1993)[13] にある.

LCMの場合, 通常は1つのサブセルに粒子を複数個入れ, ポインタもしくはLink Listを利用して考慮すべき近接粒子を特定し, 力や衝突の計算をするが, このExclusive Particle Grid Method (EPGM) は1つのサブセルに 最大1粒子だけ入れておくという戦略である.

ここでは, EPGMにおけるサブセルのことをここでは, グリッドと呼ぶことにする.

この時のグリッドの一辺の長さは,

$$\sigma < l_{gx} < \sqrt{2}\sigma \tag{7}$$

となるようにとる. (図1). 実際にプログラムを作成する場合は, 一辺に入るグリッドの数 n_{gx}, n_{gy} と長さ l_{gx}, l_{gy} を次のようにして計算すればよい.

$$l_{gx} = INT(l_x/(\sqrt{2}\sigma)) + 1 \tag{8}$$

$$l_{gy} = INT(l_y/(\sqrt{2}\sigma)) + 1 \tag{9}$$

$$n_{gx} = l_x/l_{gx} \tag{10}$$

$$n_{gy} = l_y/l_{gy} \tag{11}$$

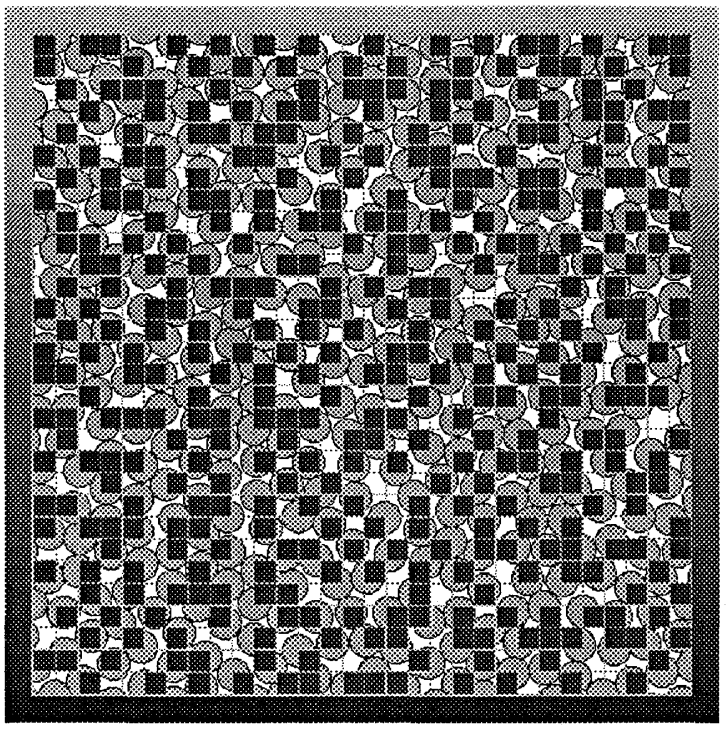

図 1: 占有率 $\nu = 0.70$ で, 実際に EEPGM を使って格子にマッピングした時の粒子配置と占有された格子

この時, グリッドの総数は, $n_g = n_{gx} \times n_{gy}$ となる.

EPGM は, 格子スピン系とのアナロジーで考えると, 隣接セルが直接粒子に 1 対 1 対応するので, N+1 準位 Potts 型正方格子に粒子をマッピングしたとも言うことができる (粒子の入っていないセルには 0 を入れるため). 粒子の連続的でランダムな配置が格子にマッピングされるため, 隣接粒子の特定や取り扱いが容易になる.

Form et al.[13] は, TSDMD の短距離相互作用する高密度のソフトコアモデル粉体系に対してこの方法適用し, ベクトル演算用に効率よく計算できるようなアルゴリズムを開発しているが, 筆者はこのアイデアを剛体円盤系の EDMD に適用し, その拡張を試みた.

まず高密度領域を考える. この場合は, EPGM のアイデアがそのまま使える. すなわち, 粒子と衝突する候補として正方形を形作る 24 個の近接格子を考慮し, それらのグリッド配列に登録されている粒子を探索し, その衝突時間を計算するだけである. この時の EPGM における考慮している粒子が所属するグリッドと隣接する 24 のグリッドが形成する部分を, 最小 (MIN.) のマスクと呼ぶことにする. (MIN. と呼ぶのは, これが EDMD をする際に考えられる最小の大きさのマスクとなるからである. これより小さいマスクでは次の節で述べる Neighbour List の時間を求める際に、考慮している粒子とマスクの外にある粒子が重なる可能性が生じるため不適当である.)

EPGM により高密度領域では, MIN. で近接グリッドには十分な数の粒子が入っており, これは衝突を考慮する粒子ペアの候補としては十分であり最適化される. よって, 本当に必要な粒子しか登録してないため近接粒子の探索の効率が増し, その結果計算効率が増大する.

次に, 低密度領域を考える. この場合, MIN. のマスク内にはほとんど衝突の候補となる粒子は登録されていないことがすぐわかる. このような状況下では, 登録粒子以外の粒子ペアが実は次の最短時間で起こるイベントに関連していたという場合もありえるため, 計算が途中で破綻する.

そこで, 筆者は EPGM のアイデアをさらに拡張することとした. 計算の破綻を防ぐためにはより大きな領域で粒子ペアの候補を探さなくてはいけない. 通常, LCM では粒子位置の大まかな登録をすませ, その後に大まかな登録を参照しながら半径 r_{NL} 内にある粒子を Neighbour List に登録するというプロセスを踏む.

しかし, EDMD の場合は連続ポテンシャルのようにきちんとした等方性を要求されるわけではないので, 候補となる粒子はおおまかな円で近似してよいことがわかる. 幸い, 既に粒子配置は格子状のグリッドにマッピングされているのでこの性質を最大限に使わない手はない.

このようなことから, 隣接粒子は格子状のグリッドを円で近似したマスクを考えればよいことがわかった. 格子を円に近似するは, コンピューターの離散空間に連続の円を描くアルゴリズム (Bresenham,Michener(1977)) が利用できる.

このようにして, R=3 から R=10 までの円を生成して表示したのが図 2 である. 図中では, MIN. もあわせて表示してある.

隣接グリッド (マスク) の総数は, それぞれ MIN.(24), R=3(36), 4(68), 5(96), 6(136), 7(176), 8(224), 9(292), 10(348) となる.

このマスクを拡張するやり方を Extended Exclusive Particle Grid Method (EEPGM) と呼ぶことにする.

EEPGM は, LCM+NL に比べ単純明快である. また, LCM のようにおおざっぱな隣接粒子と言うのではなく, 必要最小限のものしかとらないように簡単に調整できる. セルの大きさをどのくらいにするか頭をなやます必要がなくなり, パラメーターの調整の困難もなく, プログラムもわかりやすく, メモリも大幅に削減でき, 演算回数も少なくてすむ.

また 8.2 節でも述べるが, EEPGM にハッシュ法を使うと, 無限に広がる系にも高速な計算が簡単に適用可能である. EEPGM には, LCM では決して実現し得ない単純さがある.

実際に EEPGM を使ってコードを作成する時は, マスクの形があらかじめわかっているため, 注目しているグリットを中心にした相対的なポインタを配列に入れておき, ポインタ情報をもとに隣接グリットを探索していけばよい.

5 Neighbour List and Dynamical Upper Time Cut-Off

前節で導入した EEPGM をイベント毎に更新して計算を進めても, すべての粒子ペアを考慮するよりははるかに効率がよいが, 大規模計算には不十分である. そこで, EEPGM を基本にして次なる効率化の戦略を考えることにする.

ここでは, Neighbour List (NL)[9] の戦略を採用する. 幸いなことに EEPGM はグリッド内に粒子が最大 1 つしか入っていない. ここで, この EEPGM の最大の特徴を利用する. すなわち, グリッドにマッピングした時点ですでに Neighbour List の登録が完了していると見なすことができる. なぜなら Neighbour List は, マスクの形のポインタを使って粒子があるのかないのかをグリッド毎に判定していけばよいからである.

これを通常の LCM+NL のやり方と比較すると, まず LCM でサブセルに分割したのち粒子を Link List に保存する. そして NL を製作する際は, 半径 r_{NL} の中に入る粒子を Link List から探しだし, NL に登録し直すという手続きを踏む. したがって, EEPGM が一回の操作で NL の登録が完了するのに対し, LCM+NL は 2 度にわたるリストの作成という複雑な手続きを踏まなくてはならず, プログラミングの際にも困難を伴う. よって, EEPGM(+NL) がプログラムの手間が省けシンプルである. これはプログラミング, デバッグ, 拡張などに余分な手間がかからないことを意味する. しかも空間を最小限に分割し, 最小の隣接しか見ていないため効率がよいことが直感的にわか

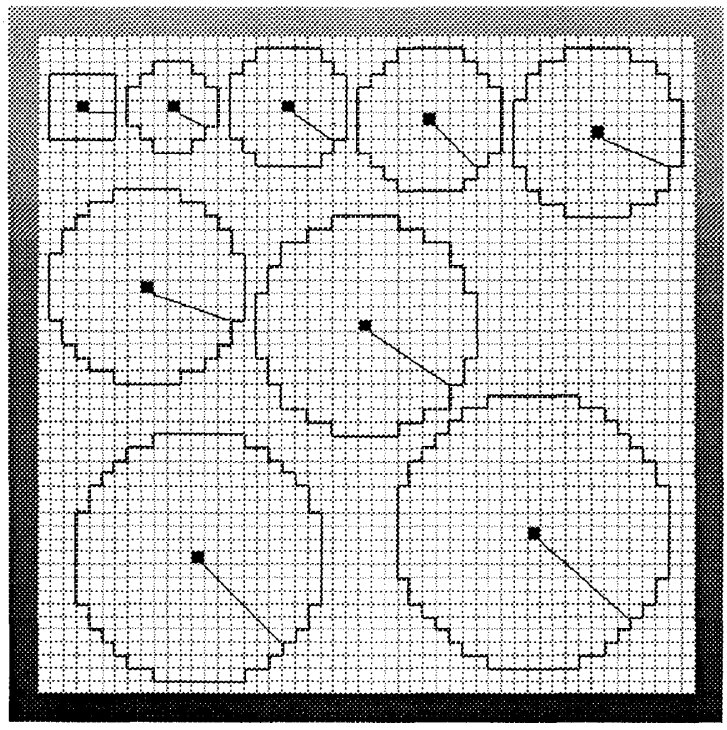

図 2: 最小隣接格子 MIN.(左上) と $R = 3 \sim 10$ までの円の発生アルゴリズムから作成した隣接格子のマスク. マスク内の実線は中心の格子の端から外枠までの最小距離を示している.

るであろう.

EEPGM で NL の登録が完了してしまったので, NL のメリットを最大限に使い, ある時間 t_{NL} 内は EEPGM のマスク内に所属する粒子のみから衝突時間を計算するとするとイベント毎の EEPGM の計算量 $\mathcal{O}(N)$ を $\mathcal{O}(1)$ に削減できる. これにより, 大幅な計算量の削減が実現する. なお, 時間 t_{NL} 経過した後は, 再び EEPGM を行って近接粒子のグリッドへの更新をしなおす.

ここでこのある時間 t_{NL} をどのようにして決定するのかを考察する. t_{NL} を決定する際は, 採用しているマスクよりも外にあるグリッドにマップされた粒子が, 注目している粒子に衝突する可能性が全くない時間でなくてはならない. この時間を適当に決めて, 実際にシミュレーションしてもたいていうまくいかない. 長すぎる t_{NL} を用いると, たった 1 つの粒子ペアのカウントミスが, シミュレーション中に負の衝突時間を生じさせ, 必ず途中で破綻する. よって, このために膨大な試行錯誤を必要とする.

そこでこれらの困難を解決するため, 以下のような手続きで t_{NL} を決める. まず EEPGM の終了した直後に, 系内の最も速い速度で運動する粒子の速度を探索しその値を保存しておく (探索にかかる計算量は, $\mathcal{O}(N)$ である). 次に注目した粒子とマスク外からの粒子が, 共に最大速度で運動しており, さらにそれらの粒子が正面衝突すると仮定したときに費やす時間 t_{min} を求める. ここで $t_{NL} = t_{min}$ とすると, t_{min} 内では系内でマスク外からの粒子が衝突するというイベントは決して起こり得ないことになる. これで衝突ペアの見落としは全くなくなり, シミュレーションが破綻することない. よって, t_{min} の間はマスク内の粒子のみを考慮しさえすればよい. t_{min} を求める際には, マスク外の領域と中心のグリッドまでの最短距離 r_{min} が必要であるが, これは採用しているマスクの幾何学からすぐにわかる (図 2). このとき最短距離 r_{min} から系内の最大粒子半径 σ_{max} の 2 倍を引いた距離を先に求めた最大速度 v_{max} の 2 倍で割れば t_{min} が求まることになる.

	MIN.	R=3	R=4	R=5	R=6	R=7	R=8	R=9	R=10
最短距離 r_{min}/l_{gx}	2	$\sqrt{5}$	$\sqrt{13}$	$\sqrt{18}$	$\sqrt{29}$	$\sqrt{40}$	$\sqrt{52}$	$\sqrt{72}$	$\sqrt{85}$

表 1: 各マスクにおける最短距離

$$t_{min} = \frac{r_{min} - (2\sigma_{max})}{2v_{max}} \tag{12}$$

平衡系においては, t_{min} はシミュレーション中に大幅に変わることはまずないのでこの戦略はうまく機能する. しかし非平衡系 (緩和過程や熱浴が存在している系) に適用しようとするととたんに破綻してしまう. なぜなら系の最大速度を持つ粒子が時々刻々と変化するからである. しかしこれは, エネルギーの増加をともなうイベントごとに最大速度のチェックをし, その都度 t_{min} を更新していくことで解決することが可能である. このチェックにかかる計算量は $\mathcal{O}(1)$ である. すなわち, 非平衡系シミュレーション中には t_{NL} は次々と変わることとなる. これらの一連の手法を Dynamical Upper Time Cut-off (DUTC) と呼ぶことにする.

この DUTC の開発により, EEPGM(+NL) は非平衡系シミュレーションも適用可能となった.

なおマスクの大きさであるが, 高密度の場合は, MIN. でもかなりの長時間 EEPGM を更新せずにすむが, 低密度の場合はダイナミクス自体が速く進むため, マスクが小さいと下手をするとイベント毎に EEPGM を更新しなくてはいけなくなる. よって, これを回避するため低密度ではより大きなマスクを使って EEPGM の更新する回数を減らす.

各マスクで中心に属する粒子とマスクの外の粒子の最短距離 r_{min} を, 表 1 に示す.

6　計算量の解析

計算量は, アルゴリズムの性能を計る重要な尺度である. よって, この節では今回開発した方法とこれまでの方法の計算量の解析をし比較検討してみる.

Rapaport 系列と筆者のアルゴリズムの最大の違いは, Cell-Crossing Event を考えるかそれとも Neighbour List の概念を使うかということに集約される. そこでこの点に特に注目し, 実際に $A \times N$ の粒子間衝突を計算する場合を考え, 両者の計算量を定数係数 (k で示す) つきで見積もることにする. なお, N はかなり大きな数だとする.

また, 両者とも高速化の手法として, Single-Event Time List, CBT, DSA を使っているとする.

- Rapaport 系列のアルゴリズム (LCM + Cell-Crossing)
  - 初期ステップと最終ステップ (×1)
    LCM — $k_{LCM} \times N$
    イベントの計算 — $k_{PP} \times 9 \times N_c \times N + k_{PC} \times 4 \times N$
    CBT の製作 — $k_{CBT} \times N$
    粒子の最終状態を得る — $k_{UPDATE} \times N$
  - 計算ステップ (loop)
    $(A \times N) \times (k_{PP} \times 9 \times N_c + k_{CBT} \log N) + (B \times N) \times (k_{PC} \times 3 + k_{CBT} \times \log N)$

N_c は, LCM により登録された 1 セルあたりに含まれる粒子の個数. ここで注意したいのは, Cell-Crossing Event は, 粒子衝突のイベントの何割かの頻度で起こるということである. よって, $A \times N$ の衝突を計算したいときは, $B \times N$ の計算すなわち, かならず $\mathcal{O}(N)$ の Cell-Crossing Event を計算しなくてはならず, これは無視できない.

- 筆者のアルゴリズム (EEPGM + DUTC)
  - EEPGM 更新ステップ ($\times C$)
    EEPGM — $k_{EEPGM} \times N$
    イベントの計算 — $k_{PP} \times N_g \times N$
    最大速度の検索 — $k_{MVS} \times N$
    CBT — $k_{CBT} \times N$
    粒子配置の更新 — $k_{UPDATE} \times N$,
  - 計算ステップ (loop)
    $(A \times N) \times (k_{PP} \times N_g + k_{CBT} \times \log N)$

となる. N_g は, 採用したマスク内に存在する粒子数の平均値で, C は EEPGM の頻度で $C \sim A$ のオーダーの値である.
両者の係数を粒子数 N のオーダーでまとめて比較すると,

- Rapaport 系列

  $(k_{LCM} + k_{PP} \times 9 \times N_c \times (A+1) + k_{PC} \times (3 \times B + 1) + k_{CBT} + k_{UPDATE}) \times N + (A \times k_{CBT} + B \times k_{CBT}) \times N \times \log N$

- EEPGM+DUTC

  $(k_{EEPGM} + k_{PP} \times 2 \times N_g + k_{MVS} + k_{CBT} + k_{UPDATE}) \times C \times N + (A \times k_{CBT}) \times N \times \log N$

注目すべきは, Rapaport 系列では Cell-Crossing Event にかかる計算量が, $B \times k_{CBT} \times N \times \log N$ と $\mathcal{O}(N \log N)$ に依存するということである.
これは, 非常に大きな粒子系では, EEPGM+DUTC の戦略の方が効率がよいことを示している.
一方, 比較的小さな粒子系では, $C \times N$ に関連する EEPGM の定数係数が Rapaport 系列のそれより大きいという可能性もありうる. 実際にはこの違いはわずかであると思われ, 定数を正確に見積もるのは困難であるため, どちらが決定的に速いといった議論はここでは不可能である. ちなみに筆者の見積もった範囲では, N の項の定数係数は, N_c と N_g の比率に強く依存し, $N_g/N_c \sim 4$ 程度で大体等しくなるようである. また, Cell-Crossing にかかる計算は粒子数増大に対しても実際には, シミュレーションが不可能なくらい相当大きな粒子系でないと違いは顕著にならないようである.

7 実際の数値シミュレーション

実際にシミュレーションをする際には計算量のオーダーはさほどあてにならないのが現実である. なぜなら粒子数が小さい時には, 定数係数に強く依存するからである. またシミュレーションの速さは, 使用する計算機の能力による機種依存性, 使用言語とコンパイラの性能, プログラマのコーディングのよしあしで効率性は大きく変化する.

著者	年代	計算機	粒子数	効率性 (Million of Collisions/CPU Hour)
Alder et al.[3]	1959	IBM704	100	0.002
小野ら [48]	1978	HITAC8800/8700	108	0.409
Rapaport[15]	1980	IBM370/168	256	1.060
内藤 [49]	1984	HITAC-S810	10976	4.04
Rapaport[29]	1988	IBM4381	14160	2.0
Lubachevsky[16]	1991	VAX8550	-	1.62
Rapaport[30]	1991	IBM3090E	57600	7.0
Marín et al.[19]	1993	SUN690	2500	16.07
Smith et al.[22]	1997	DEC300/400	100	18.0
礒部	1998	Alpha500 互換機	2500	48.98

表 2: 過去のコードとその効率性

過去に作成されたコードとの完全な比較は到底実現不可能であるが, 慣例的に作成されたコードが手持ちの計算機を使って, 1 時間当たりにどれだけの粒子衝突を計算する能力があるのかが過去の文献で紹介されており, この指標を使ってできるだけ多くの文献を集めて, それぞれのコードの効率性を比較してみることにした. (表 2, 図 3 左)

なお, 系の密度によっても効率性は全く異なってくるので, 文献に記述されている最も高い数字のみを掲載することにした.

表 2 で Smith et al.(1997) は, 3 次元ポリマー系のシミュレーションでアルゴリズムが違うので, 筆者のコードとの単純な比較はできない. Marín et al.(1993) は, 剛体円盤系でのシミュレーションを行っており, しかも, Marín et al. はその論文の中で過去に発表された主要な 2 つの高速アルゴリズム, Rapaport(1980) と Lubachevsky(1991) を基にしてコードを作成しており, それと彼らの提案したアルゴリズムを基にしたコードの直接の比較をしている. その結果, すべての密度領域で彼らのコードの効率性が勝っていることを示している. よって, 筆者が開発したコードは Marín et al. の文献を参照して比較するのが妥当であると思われ, 粒子数を等しくして計算してみた. 効率を比較するとマシンの性能を差し引いてみても高速であり, 筆者のコードは高いレベルの効率性を実現していることがわかった. (ちなみに, 粒子数 N=100 では, 効率は 82.58 でありアナライザーの解析によるとその 44% が計算とは関係ないデバイスや出力関連のルーチンに時間を費やしており, これは単純な実行時間のみで純粋なプログラムの効率を計算すること自体, もはや限界に近づいていることを意味する.)

今回開発したコードで, 実際に大規模計算した例を 1 つ挙げる. 図 3 右は, 非弾性剛体円盤 (粉体) 系の一様冷却過程 (Homogeneous Cooling Process: HCS) をシミュレーションし, その結果を各粒子毎に速度ベクトルを表示したスナップショットである.

非弾性剛体円盤系は平衡系のシミュレーションとは異なり, 衝突の際エネルギーの散逸が起こる. よって, 衝突則として粒子間相対速度の法線成分を反発係数 $r(=0.9)$ 倍する.

シミュレーションは, 粒子数を $N = 160,000$ とし, 粒子占有率を $\rho = 0.75$, 1 粒子あたりの衝突回数は, $C/N = 100$ とした時の結果である. この大規模 EDMD シミュレーションから, 系を一様冷却する過程においては図 3 右で見られるように自発的に渦構造が出現することがわかった。

ちなみに, この計算に要した CPU 時間は 52 分 (Alpha500 互換機, Linux, g77), メモリ容量は 15M 程度であり, 今回のコードの開発によりこのような大規模計算が研究室のワークステーショ

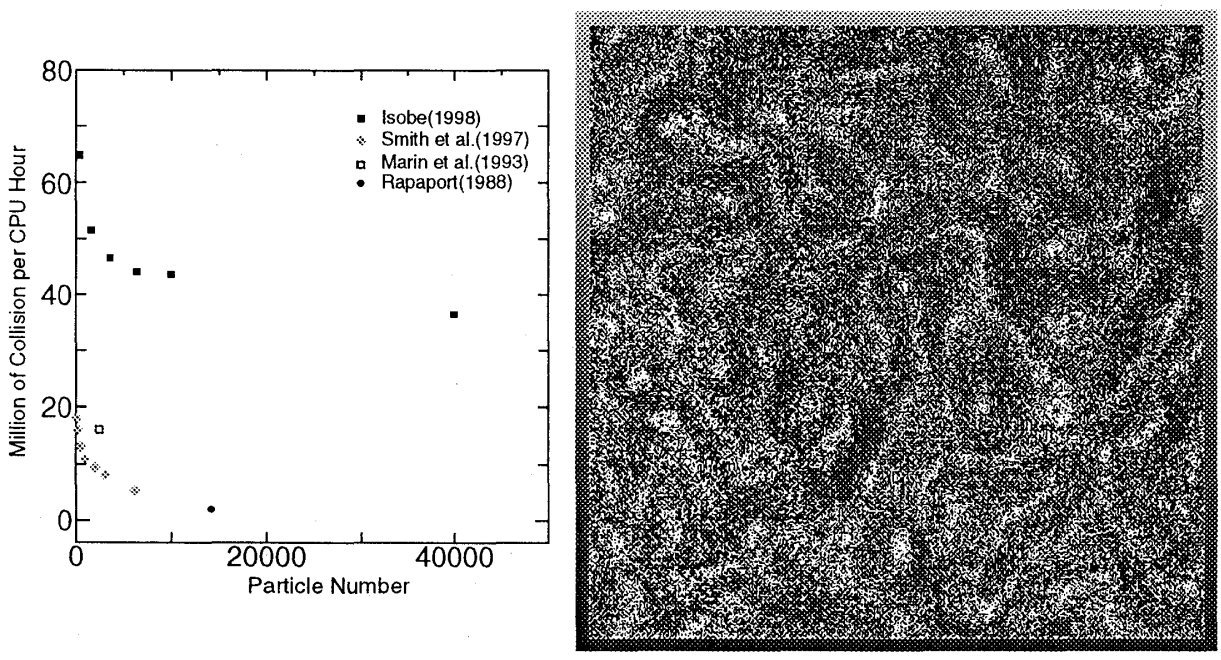

図 3: 演算効率の粒子数依存性（左）と実際のシミュレーションの例（右）

ン程度でも十分計算可能となった[3]．

8 様々な系への応用

8.1 多分散系

直径の違う様々な剛体円盤粒子が共存するような系の場合を考える．Buchholtz et al.[12] にも記述されているように，EPGM は多分散の度合いに制限があったが，EEPGM は，このような系にも制限なく容易に適用可能である．まず系の中の最も小さな粒子半径を基準にしてグリッドを作成する．分散の大きな系ではより大きな階層のマスクを使って隣接グリッドをチェックすればよい．このように，EEPGM は EPGM より広い応用範囲があるといえる．

8.2 無限に広がった系

無限に広がった系への適用例として，ここでは最も簡単な例として天井のない系（上端開放系）を考える．これは，一様な重力の下で系の底にエネルギー供給源があるような場合で，生じる状況である．

EEPGM により，系はグリッドに分割されており，グリッドには粒子 1 個しか入っていない．ところが系の上端は開放されているため，グリッドは系の上端に向かって無限個存在することになる．このことは，グリッドのための配列が無限個必要となることを意味し，シミュレーションが原理的に実行不可能となる．

そこで，必要な配列の個数を有限にとどめ，なおかつ高速なシミュレーションを行うために，デー

[3] 実際に筆者の研究室のワークステーションでも，250 万粒子でのシミュレーションが可能であった

タ探索でよく使われるハッシュ法を用いることを提案し，その適用を試みることにする．

- データの登録.

  系の横の総グリッド数を N_{gx} とする．また，横のグリッドの index を $i_g(1 \leq i_g \leq N_{gx})$ とし，縦のグリッドの index を $j_g(1 \leq j_g \leq \infty)$ とする．ここでグリッドの通し番号 $N_G(i_g, j_g)$ を，

$$N_G = N_{gx}(j_g - 1) + i_g \tag{13}$$

  とし，N_G で粒子の入っているグリッドの最大の値を N_{Gmax} とおくことにする．N_{Gmax} は，粒子のあるグリッドで最大の j_{gmax} とその時の最大の i_{gmax} から組 (i_{gmax}, j_{gmax}) より，$N_{Gmax} = N_{gx}(j_{gmax} - 1) + i_{gmax}$ がわかる．

  $1 \leq N_G \leq N_{Gmax}$ は，グリッドの 1 次元配列そのものであり，配列内には粒子番号か 0 の数列が並んでいる．これを，1 次元 *Virtual Array* と呼ぶ．この 1 次元 *Virtual Array* は，j_{max} が大きな数値である場合，配列内に大量の 0 が含まれる．この 0 は単に粒子が入っていないという情報しかないにもかかわらず，直接メモリ使用量に関連してくるものなので，ここをなんとかしたい．そこで，*Vritural Array* の情報を圧縮をすることとする．結局，必要な情報は粒子番号とその粒子がいるグリッドの index がわかってさえいればよい．よって，$A(N), BX(N), BY(N)$ という 1 次元整数配列を用意して 0 のグリッドを無視して端から順につめて $A(N)$ には粒子番号を，$BX(N), BY(N)$ にはその時のグリッドの index i_g と j_g をそれぞれ保存しておけばよいだけである．グリッド (i'_g, j'_g) に粒子があるのかないのかを知りたい時は，$BX(N), BY(N)$ を頼りに (i'_g, j'_g) に一致するものがあるかどうかを線形探索すればよい．ところが，これでは探索の際に $\mathcal{O}(N)$ の演算が必要なので非効率的である．

  そこで高速な探索を実現させるため，探索を $\mathcal{O}(1)$ で実現できるアルゴリズムとして知られるハッシュ法を使うことにする．様々なハッシュ関数が考えられ依然として開発の余地を残すが，ここでは以下のような最も単純なハッシュ関数を例にとって説明する [52].

  まずハッシュ関数は，

$$k = [\frac{N_G - 1}{L}] + 1 \tag{14}$$

  として *Virtual Array* を等分割し，通し番号 N_G に対してキー k をそれぞれ求める．ここで L は分割する際の *Virtual Array* の 1 部分の大きさである（例えば，5 ～ 10）．先に求めた N_{Gmax} より，$k_{max} = [\frac{N_{Gmax}}{L}]$ が決まる．ここで，新たに配列 $C(k_{max}), D(k_{max})$ を用意し，*Virtual Array* を L で分割した際，k がそれぞれ $A(N)$ のどこから始まるのかを $C(k)$ に，またそれはどのくらいの大きさであるかを $D(k)$ に記録しておく．

  この場合，実際に必要な配列は，$A(N), BX(N), BY(N), C(k_{max}), D(k_{max})$ であり有限に留まる．またハッシュ法を用いるために使う新たな配列は，$C(k_{max}), D(k_{max})$ のみであり，$k_{max} < N$ であるので，メモリ使用量は線形探索の場合よりもわずかに増加するだけである．

- 探索の過程

  さて，ここでグリッド i'_g, j'_g に粒子は入っているのか？入っているとしたらその粒子番号は何番か？ということが知りたいとする．まず，$N'_G = N_{gx}(j'_g - 1) + i'_g$ により，通し番号を計算する．次に，先のハッシュ関数 (14) 式を使って k' を求める．この k' により，$C(k'), D(k')$ から BX, BY の探索範囲が $[C(k') \sim C(k') + D(k') - 1]$ $(\leq L)$ の index のみに限定される．

BX, BY の探索の結果, グリッド (i'_g, j'_g) に等しいものがあればそのときの BX, BY の index s より $A(s)$ がわかり, その $A(s)$ がグリッド (i'_g, j'_g) に入っている粒子番号である. また探索の結果, 一致したものが見つからなければグリッドには粒子は入っていないことになる.

この手続きをふめば, 高速シミュレーションは可能となる. 線形探索が計算量 $\mathcal{O}(N)$ であったのに対し, ハッシュ法では探索は L の長さにしかよらないため計算量が $\mathcal{O}(1)$ となるからである.

他の境界条件についても一端1次元の *Virtual Array* を製作してしまえば後は基本的に同じ手続きで計算可能である. 従って, 原理的にはどのような境界であろうと高速シミュレーションは可能であると思われる. ハッシュ関数は今回は最も簡単な等分配するものを用いたが, 粒子の入っているグリッドが偏っている場合などは明らかに不適当であるので改良の余地があるだろう. しかし, これはこれからの課題である.

このように, グリッド登録を基本としたEEPGMの戦略は, 拡張が簡単であるというメリットがある. アルゴリズム的に発展性があるということもアルゴリズムの評価の重要なポイントである.

8.3 低密度系

低密度系における問題点は, グリッドに割り当てる配列が粒子数に比べ圧倒的に多く膨大になるという点である. これはメモリ容量の面から言って好ましくない. しかし, 8.2 節と同様にグリッド配列を粒子数の大きさに圧縮し, ハッシュ法を用い補助的な配列を作成して近接グリッドの情報を高速に引き出せば, 効率性, メモリ使用量の点でも問題ない.

8.4 3次元系

もちろん3次元への拡張も容易である. この場合, 隣接グリッドの数が2次元の場合と比べ多くなるのでその分効率は落ちるであろう.

8.5 Time-Step Driven 型分子動力学系

例えば, 粒子半径 σ の数倍の半径でポテンシャルのカットオフを設けたソフトコアポテンシャル系のシミュレーションなどへの適用を考える.

EEPGMの概念を使えば中心の粒子を決めたときの隣接する格子を形作るマスクは円を基本としており、隣接粒子の等方性を満足する. よって, Neighbour Listを製作する際のカットオフ半径に対応するRを選んで, EEPGMで粒子配置を格子にマッピングし, マッピング情報を下に力のカットオフ半径に存在する粒子ペアを割り出し, 力を計算することが可能であり, これはLCM+NLの戦略よりも登録の過程が単純化され, しかも高速なシミュレーションが可能となると考えられる.

9 まとめ

本論文では, プログラムが簡潔でわかりやすく, なおかつ大規模計算に適した高速な剛体円盤系のアルゴリズムを提案し, その紹介をした. このアルゴリズムは, 既存の文献の効率性と比較しても高速であり, 世界最高速のレベルで効率よく粒子衝突を計算をする能力を持っていることがわ

かった. また, 様々な系への応用も簡単に実現できることから汎用性もあり, アルゴリズムの評価の指標である, 単純性, 効率性, 汎用性の３つを満足するものであることがわかった.

計算するイベント数が同じであれば, 粒子数増大による計算量の増加は, $\log N$ 程度で極めてゆるやかなものであり, 研究室のワークステーション程度 (1998 年現在) でも, 数百万粒子 ($\sim 10^6$) 程度であれば計算可能である. また, メモリーとハードディスク容量が許すならさらなる超大規模シミュレーションももはや夢ではない.

剛体円盤 MD は理論的取り扱いの基礎となり, 大きな粒子系で長時間のシミュレーションが可能になれば応用範囲が極めて広く, 理論の検証など様々な分野における研究にはかかせないツールといえよう.

最後に, 剛体円盤系とソフトコア系との違いについて言及する.

ソフトコア系での TSDMD では、LCM + NL の戦略を使い高速なシミュレーションをすることは手法として確立している. そこで, これらの高速化の手法をフルに使った, 剛体円盤系とソフトコア系ではどちらのシミュレーションがより高速に系のダイナミクスを計算できるか, またそれらの系の特徴的な違いはなにかという疑問がわく.

この問いに答えるため, Rapaport[31] はレイリーベナール対流系において, 実際に両者のプログラムを走らせ, その効率性を比較してみせた. その結果 3-4 倍程度剛体円盤系の方が, ダイナミクスに対する知見を速く得るという事実を報告している. また, Smith et al.[46] の論文では, ポリマー系では 30-50 倍も短い計算で同じダイナミクスが実現できるという報告もある.

剛体円盤系とソフトコア系のダイナミクスに質的な違いがあるかどうかの系統的な研究はおそらくまだなされていないと思われる. というのもこれまでのシミュレーションは, 粒子数が少ない場合であったため, その違いなど誤差の範囲に隠れてしまっていた可能性があり, よく分からなかったというのが現状であろう.

その意味で, 今回紹介したアルゴリズムなどを使えば大規模計算が手軽に実現できるようになり, ようやく, これらの本質的な違いを議論できる研究が可能となる時代に入ったと言える.

なお, 今回提案したアルゴリズムは, スカラーマシンでの最も威力を発揮するもので, 並列計算には対応してない. 並列計算用アルゴリズムの開発はこれからの課題である.

謝辞

本論文をまとめるにあたって, 内容に関して議論並びに原稿を通読していただいた中西秀教授に感謝します. またアルゴリズム開発に当たって随時議論していただいた愛知工業大学の村中正氏にも謝辞を表します.

略語のまとめ

EDMD — Event-Driven 型分子動力学法

TSDMD — Time-Step-Driven 型分子動力学法

BST — Binary Search Tree (2 分探索木)

CBT — Complete Binary Tree (完全 2 分探索木)

LCM — Linked Cell Method

NL — Neighbour List

LMA — Local Minima Algorithm

DSA — Delayed State Algorithm

(E)EPGM — (Extended) Exclusive Particle Grid Method

DUTC — Dynamical Upper Time Cut-off

参考文献

[1] B. J. Alder and T. E. Wainright, in *International Symposium on Statistical Mechanical Theory of Transport Processes, Brussels 1956*, edited by I.Prigogine (Interscience, New York, N.Y., 1958).

[2] B. J. Alder and T. E. Wainright, *J. Chem. Phys.* **27** (1957), 1208.

[3] B. J. Alder and T. E. Wainright, *J. Chem. Phys.* **31** (1959), 459.

[4] B. J. Alder and T. E. Wainright, *J. Chem. Phys.* **33** (1960), 1439.

[5] B. J. Alder and T. E. Wainright, *Phys. Rev.* **127** (1962), 359.

[6] B. J. Alder, D. M. Gass, and T. E. Wainright, *J. Chem. Phys.* **53** (1970), 3813.

[7] B. J. Alder, *Annu. Rev. Phys. Chem.* **24** (1973), 325.

[8] J. J. Erpenbeck and W. W. Wood, in *Modern Theoretical Chemistry Vol.6, Statistical Mechanics Part B*, edited by B.J.Berne (Plenum, New York, 1977), Chap. 1, p. 1.

[9] L. Verlet, *Phys. Rev.* **159** (1967), 98.

[10] B. Quentrec and C. Brot, *J. Comput. Phys.* **13** (1975), 430.

[11] R. W. Hockney and J. W. Eastwood, *Computer Simulation Using Particles* (McGraw-Hill, New York, 1981).

[12] V. Buchholtz and T. Pöschel, *Int. J. Mod. Phys. C* **4** (1993), 1049.

[13] W. Form, N. Ito, and G. A. Kohring, *Int. J. Mod. Phys. C* **4** (1993), 1085.

[14] M. P. Allen and D. J. Tildesley, *Computer Simulation of Liquids* (Clarendon Press, Oxford, 1987).

[15] D. C. Rapaport, *J. Comput. Phys.* **34** (1980), 184.

[16] B. D. Lubachevsky, *J. Comput. Phys.* **94** (1991), 255.

[17] K. Shida and Y. Anzai, *Comput. Phys. Commun.* **69** (1992), 317.

[18] K. Shida and Shin'ichi Yamada, *Comput. Phys. Commun.* **86** (1995), 289.

[19] M. Marín, D. Risso, and P. Cordero, *J. Comput. Phys.* **109** (1993), 306.

[20] M. Marín and P. Cordero, *Comput. Phys. Commun.* **92** (1995), 214.

[21] A. T. Krantz, *ACM Trans. Model. Comput. Sim.* **6** (1996), 185.

[22] S. W. Smith, C. K. Hall, and B. D. Freeman, *J. Comput. Phys.* **134** (1997), 16.

[23] D. E. Knuth, *Sorting and Searching, The Art of Computer Programming*, Vol.3, (Addison-Wesley, Reading, Mass., 1973).

[24] M. Mareschal and E. Kestemont, *Phys. Rev. A* **30** (1984), 1158.

[25] E. Meilburg, *Phys. Fluids* **29** (1986), 3107.

[26] M. Mareschal and E. Kestemont, *J. Stat. Phys.* **48** (1987), 1187.

[27] M. Mareschal, M. Malek-Mansour, A. Puhl, and E. Kestemont, *Phys. Rev. Lett.* **41** (1988), 2550.

[28] A. Puhl, M. Malek-Mansour, and M. Mareschal, *Phys. Rev. A* **40** (1989), 1999.

[29] D. C. Rapaport, *Phys. Rev. Lett.* **60** (1988), 2480.

[30] D. C. Rapaport, *Phys. Rev. A* **43** (1991), 7046.

[31] D. C. Rapaport, *Phys. Rev. A* **46** (1992), 1971.

[32] I. Goldhirsch and G. Zanetti, *Phys. Rev. Lett.* **70** (1993), 1619.

[33] S. McNamara and W. R. Young, *Phys. Rev. E* **53** (1996), 5089.

[34] I. Goldhirsch and M-L. Tan, *Phys.Fluids* **8** (1996), 1752.

[35] M-L. Tan and I. Goldhirsch, *Phys. Fluids.* **9** (1997), 856.

[36] F. Spahn, U. Schwarz, and J. Kurths, *Phys. Rev. Lett.* **78** (1997), 1596.

[37] P. Deltour and J. -L. Barrat, *J. Phys. I France* **7** (1997), 137.

[38] S. McNamara and W. R. Young, *Phys. Rev. E* **50** (1994), R28.

[39] S. McNamara and J. -L. Barrat, *Phys. Rev. E* **55** (1997), 7767.

[40] S. E. Esipov and T. Pöschel, *J. Stat. Phys.* **86** (1997), 1385.

[41] C. Bizon, M. D. Shattuck, J. B. Swift, W. D. McCormick, and H. L. Swinney, *Phy. Rev. Lett.* **80** (1998), 57.

[42] J. R. de Bruyn, C. Bizon, M. D. Shattuck, D. Coldman, J. B. Swift, and H. L. Swinney, *Phy. Rev. Lett.* **81** (1998), 1421.

[43] M. D. Rintoul and S. Torquato *J. Chem. Phys.* **105** (1996), 9258.

[44] B. D. Lubachevsky and F. H. Stillinger, *J. Stat. Phys.* **60** (1990), 561.

[45] B. D. Lubachevsky, F. H. Stillinger, and E. N. Pinson, *J. Stat. Phys.* **64** (1991), 501.

[46] S. W. Smith, C. K. Hall, and B. D. Freeman, *J. Chem. Phys.* **102** (1995), 1057.

[47] 和田三樹, 巨勢　朗, 戸田盛和:「融解現象とラテックス粒子による結晶模型 —セミマクロな秩序・無秩序転移」, 科学 Vol.42 No.12 (1972), 646.

[48] 小野　周, 内藤豊昭, 仁木　清:「剛体球流体の非平衡状態—分子動力学による研究— 」, 科学 Vol.48, No.11 (1978), 681.

[49] 内藤豊昭:「分子動力学法による剛体球系のシミュレーション」, 数理科学 NO.251, MAY, (1984), 32.

[50] 川勝年洋:「化学反応をともなう凝縮系の計算機実験」,　計算物理学と計算化学：田中　実, 山本良一編 海文堂 (1988).12 章.

[51] 志田晃一郎, 川合敏雄:「非弾性粒子が作るクラスター」, 物性研究 58-4 (1992), 337.

[52] 高田健次郎:「数値計算にハッシュ法を応用する」, 九州大学大型計算機センター広報 Vol.28, No.2 (1995), p129.